View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Nottingham ePrints

Chen, Jinfu and Wang, Huanhuan and Towey, Dave and Mao, Chengying and Huang, Rubing and Zhan, Yongzhao (2014) Worst-input mutation approach to web services vulnerability testing based on SOAP messages. Tsinghua Science and Technology, 19 (5). pp. 429-441. ISSN 1007-0214

Access from the University of Nottingham repository:
http://eprints.nottingham.ac.uk/51840/1/Web%20services%20testing%20paper%282014.04.01.Final%20revision%29.pdf

Copyright and reuse:

The Nottingham ePrints service makes this work by researchers of the University of Nottingham available open access under the following conditions.

This article is made available under the Creative Commons Attribution licence and may be reused according to the conditions of the licence. For more details see: http://creativecommons.org/licenses/by/2.5/

A note on versions:

The version presented here may differ from the published version or from the version of record. If you wish to cite this item you are advised to consult the publisher's version. Please see the repository url above for details on accessing the published version and note that access may require a subscription.

For more information, please contact eprints@nottingham.ac.uk

Worst-input Mutation Approach to Web Services Vulnerability Testing based on SOAP Messages

Jinfu Chen[1,2 +], Huanhuan Wang [1], Dave Towey [3], Chengying Mao [2,4], Rubing Huang [1,5], Yongzhao Zhan [1]

1 (School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, 212013, China)
2(Faculty of Information and Communication Technologies, Swinburne University of Technology, Hawthorn, Victoria 3122, Australia)
3 (School of Computer Science, The University of Nottingham Ningbo China, Ningbo, 315100, China)
4 (School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang, 330013, China)
5 (School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China)

+Corresponding email: jinfuchen@ujs.edu.cn

Abstract—The growing popularity and application of Web services have led to an increase in attention to the vulnerability of software based on these services. Vulnerability testing examines the trustworthiness, and reduces the security risks of software systems, however such testing of Web services has become increasing challenging due to the cross-platform and heterogeneous characteristics of their deployment. This paper proposes a worst-input mutation approach for testing Web service vulnerability based on SOAP (Simple Object Access Protocol) messages. Based on characteristics of the SOAP messages, the proposed approach uses the farthest neighbor concept to guide generation of the test suite. The test case generation algorithm is presented, and a prototype Web service vulnerability testing tool described. The tool was applied to the testing of Web services on the Internet, with experimental results indicating that the proposed approach, which found more vulnerability faults than other related approaches, is both practical and effective.

Keywords-*Web service vulnerability; SOAP message; Test case generation; Mutation operator; Security testing*

I. INTRODUCTION

Due to the rapid development and wide application of the Internet, use of the service-oriented architecture (SOA) for distributed Web systems has been increasing. Although Web services are the typical form of SOA, and have been the focus of widespread attention and application, their quality and reliability problems represent significant obstacles to further development. Furthermore, due to some Web service characteristics, traditional software testing approaches are not easily applied. Some factors that contribute to the difficulty in application include: (1) different development and application environments (which increases the testing difficulty before the Web services are deployed); (2) the characteristics of Web service distribution, discovery, and dynamic bindings, as well as the uncertain and invisible processes; and (3) the need for a service interface for the Web service design and implementation when applying automatic testing methods and techniques.

Although the testing of Web service robustness has already been examined [1-4], and a number of tools proposed, several difficulties and shortcomings remain, including: (1) a need for significant intervention in the testing process; (2) that only simple performance and access testing have been performed; and (3) that the approaches used in SOAP (Simple Object Access Protocol) message mutations are not optimal, with most studies to date being based on Web Services Definition Language (WSDL) specifications and Extensible Markup Language (XML) documents, and few using SOAP messages. A Web service, whose structure and source codes are not visible to the client, is located on the service provider's site, making research into its vulnerability challenging. Web service vulnerability refers to flaws in the service which threaten the security of the computer system, for example, memory leaks, buffer overflows and cross-boundary access (where memory variables access areas outside their defined scope). Some types of Web service vulnerability faults might not be effectively revealed by traditional approaches, including memory security faults, which are often triggered by illegal parameter values; and arithmetic security faults, which are often caused by parameter interaction such as dividing by zero, and out-of-range operand values.

To address the issue of testing Web service vulnerability, we propose an approach based on SOAP message mutation and the worst-input technique. The worst-input mutation method, which uses characteristics of SOAP messages, is presented in detail in this paper. The corresponding automatic test case generation algorithm, namely the test case generation based on the farthest neighbor (TCFN), is also discussed. The method involves partitioning the input domain into sub-domains according to the number and type of SOAP message parameters in the TCFN, and then selecting the candidate test case whose distance is farthest from all executed test cases and applying it to test the Web service. Finally, a prototype Web service vulnerability testing tool is implemented and applied to a number of real Web services, with experimental results showing that the proposed approaches are both effective and practical.

The main contributions of this paper are as follows:

- We propose a set of mutation operators which can automatically mutate Web service SOAP messages based on security rules and message parameter types.
- Using the farthest neighbor concept, we propose a worst-input mutation method to test Web service vulnerability, and present test case generation algorithms based on the number and type of SOAP message parameters.
- We implement the proposed approach in a Web service vulnerability testing system (WSVTS) tool, which we further evaluate through comparison with other Web service testing approaches. The results show that in most cases the proposed approach can detect more faults.

The remainder of the paper is organized as follows: some related Web service testing work is discussed in Section II. The mutation operators and security rules are presented in Section III. The details of the proposed approach are presented in Section IV, with some experiments to evaluate it reported on in Section V. The future work and conclusion are given in Section VI.

II. Related Work

Currently, research into Web service vulnerability testing remains limited, with studies focusing mainly on functionality testing [2,5,6], reliability analysis [3], data perturbation [7-9], and Web service rule mutation [10-12].

Takase & Tajima [2] proposed an approach to the functional testing of Web services by first extracting the SOAP message using the WSDL converter, and then exchanging messages using the SOAP message binding framework. A disadvantage of this approach, however, is that it only bundled some of the input parameters to obtain the return value for a single message, rather than bundling multiple interdependent functions. If the combined services could be processed on the physical machine at the same time, then the process could be more efficient. Sun et al. [5,6] have proposed a metamorphic relations-based approach to testing Web services in the context of SOA without the need for oracles. An alternative approach based on fault injection was proposed by Wu et al. [3], but the working mode of SOAP documents could not be tested; multiple mistakes could not be injected at the network layer; and the fault injection messages could not be authenticated. An approach based on data communication perturbation was proposed by Almeida & Vergilio [7], where the perturbation operators were designed according to characteristics of the SOAP message. Experiments were conducted using their proposed mutation operators and SMAT-WS [7] tools, but it was found that the designed mutation operators were not sufficient for comprehensive testing. Fuzzy approaches to generating perturbation test cases have also been studied [8, 9], but to date, an appropriately feasible test case generation algorithm has still not yet been presented.

Web service data value perturbation and rule mutation are the focus of the current paper. An approach to test-case generation based on data value perturbation was proposed by Offutt & Xu [10], where request messages were modified by mutation operations resulting from data value perturbation, RPC (Remote Procedure Call) communication perturbation, and data communication perturbation. However, only some special values such as maximum and minimum, and valid decimal, were considered in the mutation process. Their data value and communication perturbation approach [10] was modified by de Melo & Silveira [11], who also extended the mutations [12] introduced previously [1, 7], using an invalid test case value in the data value perturbation, and introducing two strategies (*all* and *choice*) and four mutation operators for RPC communication in the data communication perturbation. The test coverage for the RPC and document communication was also increased, but the overall mutation testing approach was not completely comprehensive, nor was a test case generation algorithm proposed.

We previously proposed a combinatorial mutation approach for testing the interactive faults of Web services [13]. The proposed approach defines the corresponding combinatorial strategies based on SOAP message mutation and combinatorial testing, allowing multiple mutants to be injected at one time to help uncover interactive faults. However, if the tested Web services have only one service method or one method parameter, then the combinatorial mutation approach cannot offer its full potential advantage. In order to test different kinds of Web services, we propose a worst-input mutation method based on the farthest neighbor concept, which, as a complementary approach to combinatorial mutation, can also enhance the effectiveness of Web service vulnerability detection.

III. MUTATION OPERATORS AND SECURITY RULES

The appropriate design of mutation operators is critical for mutation testing based on SOAP messages, and for it to be successful, the object and the purpose of mutation should be explicitly clear. SOAP is a message protocol based on an XML document, which forms the basis of the mutation object. A formal description for the XML modeling of a SOAP message was given by Novak & Zamulin [14]. Offutt & Xu [10] extended the regular tree grammar (RTG) model to $<E, N, D, P, A, n_s>$, but no specific parameter type information or classification were provided for the general characteristics of the XML document. Based on these models, we have improved and extended the RTG to an eRTG (extended regular tree grammar), which is a 6-tuple $<E, N, DT, P, A, n_s>$, where: E is a finite set of elements; N is a finite set of non-terminals; DT is a finite set of data types defined as {*int, string, bool, numerical, char, object}; P* is a finite set of production rules; n_s is the starting non-terminal; and A is a 2-tuple $<n, type>$ with n as the number of parameters, and *type* as the parameter type, one of {*rec, cir, cur*}, where: *rec* is the rectangular input domain, *cir* is the circular input domain, and *cur* is the curved input domain. Given a set of all element instances N, a mutation operator is $r = f(n_1, n_2, ..., n_i)$, where f is a function, $i \geq 1$, each $n_1, n_2, ..., n_i \in N$ and has an arbitrary data type, and r outputs the mutated $n_1,...,n_i$ with the same data type as the input $n_1,...,n_i$.

Although a set of interference operators had been introduced previously [15, 16], the uncertainty and randomness of an initial object led to data redundancy and low efficiency after mutation. We have therefore designed a total of 15 mutation operators for SOAP parameter types combined with Web services features, as shown in Table I.

Table I. Mutation operators of web service vulnerability testing based on the SOAP message

ID.	Operator	Brief description	Cases / Examples
01	SVB	Set the Value of n to be Blank	Change value *n* to " "
02	SVN	Set the Value of n to be Null	Change value *n* to *null*
03	IPO	Insert Parameter Operator into the value assigned to a node *n*	Insert absolute value symbol into the value assigned to node *n*
04	DNS	Delete a Node *n* and its child nodes from the SOAP message	Delete root nodes and child nodes from the SOAP message
05	FVS	Format the Value of String	"%n %n……(256)","%s %s(1024)"et al.
06	IIV	Integer Irregular Value	0,+/-(1,2^8-1,2^8,2^8+1,2^{16},2^{16}+1,2^{16}-1,2^{32},2^{32}+1,2^{32}-1,2^{64}, 2^{64}+1)
07	FIV	Float Irregular Value	0, 1, -1, +/-(the max float point +/-1), +/-(the min float point +/-1),5E-324,1.7E+308,pi,e
08	CIV	Char Irregular Value	'A','Z',:Null,'a','z',' ', '','../','{','(','[','\n','\0','\s','\d'
09	EOV	Exchange the Order of Values assigned to nodes	Exchange the order of the values assigned to n_1, n_2
10	EON	Exchange the Order of Nodes	Exchange the order of n_1, n_2
11	RSV	Random String Value	Escape character string"\e\n\r\d\x\s", "\xff\xfe\x00\x01\x42\xb5\nnnn\h9cc..."
12	LSV	Long String Value	Generate String(int n) such as:"AAA……(256)","AAA……(1024)","AAA…(15000)"
13	UVF	Url and the Value of File directory string	"http://ddddddeeeeerrttttt"; "//sytem32//Notepad.exe", H:\ABC\killvirus.exe","D:\AA.exeexe"\
14	SSI	SQL String Injection	"a or 1=1", "delete","drop table users","sql attempt5--"
15	PFB	Parameter Flip Bit	Use ReverseBit() to flip the value assigned to a node *n*

We defined a security rule for testing the vulnerability of Web services based on the proposed mutation operators as follows: the vulnerability of Web services is $Vws = G(r)$, where $r = f(n_1, n_2, ..., n_i)$ is the mutation operator for the tested Web service; $G(r)$ represents the vulnerability which is triggered by r; and

$n_i \in N$ are the Web service input parameters. When the tested Web services accept the input parameters, if any exceptions are triggered by the mutation operators, then the tested Web service is deemed to have some vulnerability flaws.

It is usual to encapsulate data in a SOAP protocol format, and a SOAP message can be expressed as two parts: input parameters and security control rules. Based on the SOAP message input parameters, a worst-input mutation approach to SOAP message mutation testing is proposed and presented in the following section.

IV. WORST-INPUT MUTATION APPROACH

With regular mutation [7], the mutant can be obtained through a small modification of the legitimate input. Taking the opposite perspective, we identify the farthest neighbor sequence from the legitimate input as the test data to generate test cases according to the SOAP message types. Effective test cases should have the greatest possible test coverage, typical representation for triggering faults, and low redundancy. The farthest neighbor idea is similar to the concept of adaptive random testing (ART) [17], which is based on various empirical observations showing that many program faults result in failures manifesting in contiguous areas of the input domain. Therefore suggesting that, if previously executed test cases have not revealed a failure, new test cases should be as far away from the already executed non-failure causing test cases as possible. Intuitively speaking, the farthest test cases have higher probability of detecting Web service security exceptions. Hence, we investigate some farthest algorithms to detect the security exceptions of Web services based on related ART algorithms and mutation.

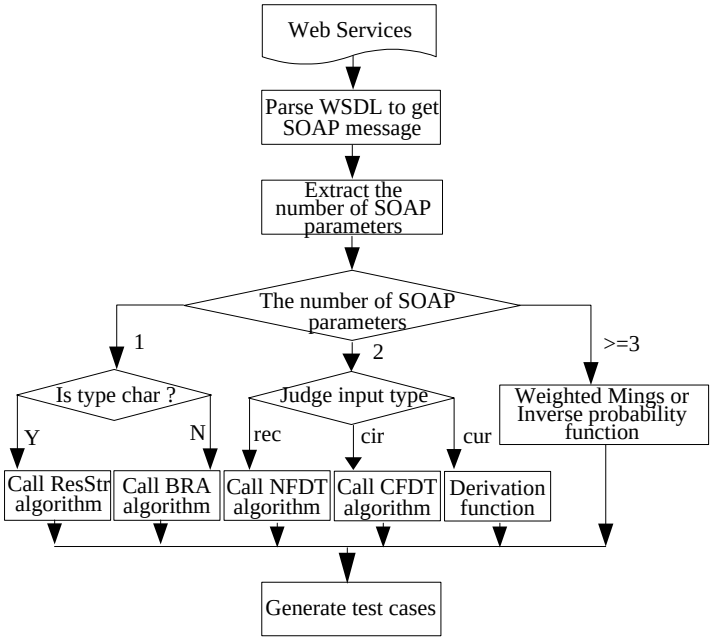

Figure 1. Flow chart of test case generation using the farthest neighbor algorithm

The input domain is partitioned into sub-domains according to the number and type of SOAP message parameters. A corresponding test case generation algorithm is then selected, and test cases conforming to the requirements of each sub-domain are then randomly generated. The candidate test case whose distance is farthest away from all executed test cases is then selected and applied to test the Web service. Here we propose the TCFN algorithm (Algorithm 1) based on the presented eRTG model. The TCFN algorithm consists of six sub-algorithms: BRA (bit reversal); ResStr (string reversal); NFDT (next furthest distance test); CFTD (circle furthest distance test); a weighted Ming distance [18]; and a multidimensional variation inverse probability distribution. BRA or ResStr are used when the SOAP message has only one parameter; NFTD or CFTD are used when there are two; and the weighted Ming distance or inverse probability distribution algorithms are used when there are more than two parameters. As can be seen in the TCFN flow chart (Figure 1), the SOAP message is obtained by parsing the WSDL file of the Web services being tested. Using an XML analysis technique, the number and type of SOAP message parameters are extracted, based on which, different algorithms are then called to generate the test cases.

Algorithm 1: TCFN
Input: the input domain D (X_{min}, Y_{min}) (X_{max}, Y_{max}) of the SOAP message parameter.
Output: the set of test cases $S=\{e_1,e_2,...,e_n\}$.

```
(1)  if (n==1) then
(2)    { if ( DT is numerical )
(3)      call BRA algorithm and related mutation operators;
(4)    if (DT is string) then
(5)      call ResStr algorithm and related mutation operators;
(6)    }
(7)  else
(8)  if (n==2) then
(9)    { divide the type of the input region according to parameter's value;
(10)   if (type==rec) then
(11)     call the NFDT Algorithm;
(12)   else
(13)   if (type==cir) then
(14)     call the CFDT Algorithm;
(15)   else
(16)   if (type==cur) then
(17)     generate the max-value and the min-value of the same interval of the function according to
           input region distribution function and related mutation operators;
(18)   }
(19) else
(20)  if ( n >=3 ) then
(21)    {
(22)      call the inverse probability distribution or weighted Ming distance algorithms based on
            parameter features;
(23) }
(24)  output the set of test cases S={e1,e2,...,en}.
```

The input region is divided into subregions based on the number and type of message parameters, and then the appropriate algorithm is selected to generate test cases to test the Web service. The main sub-algorithms of the TCFN algorithm are as follows:

(1) **BRA Algorithm**

When the input parameter data type is Integer (int), the BRA algorithm and related mutation operators are used to generate the farthest test cases. The BRA algorithm flips all bits (from 0 to 1, and 1 to 0).

(2) **ResStr Algorithm**

The ResStr algorithm calculates the length of the string, reverses it, and uses the CIV mutation operator to increase or decrease the length of the reversed string. The Web service SOAP message can be mutated using the reversed string, after which the response information of the client is examined to determine the vulnerability.

(3) **NFDT Algorithm**

The NFDT algorithm is based on the adaptive random testing (ART) family of algorithms [19]. The test cases are divided into sets E (Executed) and C (Candidate), both of which are initially empty, but as testing progresses, E contains n executed test cases $\{e_1, e_2, e_3,...,e_n\}$, and C contains k random candidate test cases $\{c_1, c_2, c_3,...,c_k\}$. ART research suggests that changes in the Candidate set size have little impact on the speed of detecting the first failure when $k \geqslant 10$, so as with previous studies, we set k to 10 in this experiment [19]. At the start of testing, when E is empty, a test case e is generated randomly, executed, and then appended to E. The next test case, c_j, can be selected from C by calculating the distance between each element of C, and the executed test case e, and then selecting that element (c_j) which has the greatest distance. The NFDT algorithm is shown in Algorithm 2.

Algorithm 2: NFDT
Input: the input domain D (X_{min},Y_{min})(X_{max},Y_{max}) of the SOAP message parameter.
Output: the set of test cases $S=\{e_1,e_2,.....,e_n\}$

(1) input the region D of soap message {(X_{min}, Y_{min}) (X_{max}, Y_{max})}
(2) set E={}, C={}.
(3) randomly generate the first test case $e(x, y)$ by using related mutation strategies and operators, and divide D into T and L by e' *x-value*.
(4) select $T\{(i, j),(s,t)\}$from D,$(e \notin T)$,$D \leftarrow D\text{-}T$;
(5) **while** (D != NULL) **do**
(6) { **if**（T!=L）
(7) { **if** (*(x-i)* $\geqslant$*(s-x)*)
(8) the next test case is generated from $T\{(i, j),(x, t)\}$, then $D=D \cup\{(i,j),(x,t)\}$;
(9) **else**
(10) the next test case is generated from $L\{(x, j),(s, t)\}$,then $D=D \cup\{(x,j),(s,t)\}$;
(11) }
(12) **else**
(13) select a field T or L randomly;
(14) select a big field $T' \in D$, and randomly generate k test cases $\{c_1,c_2,....,c_k\}$ by using related mutation strategies and operators, $C=C \cup \{c_1,c_2,...,c_k\}$.
(15) sort the set of x value from small to large;
(16) find a test case $e \in E$, find C_j by using improved binary search method , whose x-axis is nearer to e
(17) calculate the distance $d=\sqrt{\Delta x^2+\Delta y^2}$;
(18) calculate all the distances Δx_i between e and all the test cases behind C_j;
(19) **for** each C_i from C_j to C_k **do**
(20) { **if** ($d > \Delta x_i$)
(21) $d=d_{\text{new}}$
(22) **else**
(23) stop calculating according to $d < \Delta x < \sqrt{\Delta x^2+\Delta y^2}$;
(24) }
(25) **for** each C_i from C_1 to C_j **do**
(26) { **if** ($d > \Delta x_i$)
(27) $d=d_{\text{new}}$
(28) **else**
(29) stop calculating according to $d < \Delta x < \sqrt{\Delta x^2+\Delta y^2}$;
(30) }
(31) search the max value d corresponding test case C_j as the next test case, $C_j \rightarrow e$ and $E=E \cup e$, the two fields divided by C_j are joined in D;
(32) $D \leftarrow D\text{-}T$;
(33) }
(34) **return** the set of test cases $S=\{e_1,e_2,...,e_n\}$.

The original binary search algorithm [20] is improved in step 13 to increase the search efficiency and to verify its effectiveness. Since the input region is finite set, as the number of test cases grows, so too does their density in the corresponding input region – the distance between a new test case and the nearest executed test case becomes much smaller. The candidate test cases can be considered when the distance between test cases (d) is relatively large. A ratio parameter is then defined on the basis of the binary search algorithm as follows: an *array* [N] is an ordered integer array whose values range from small to large, and the sub-array from *array* [L] to *array* [H] is one sub-array of the ordered array, and the element *array* [*mid*] is the value which is the nearest to target value x. The *mid* is then selected. Hence, the ratio parameter formula is *R=(x-array[L])/(array[H]-array[L])*, then the formula *(x-array[L])/(array[H]-array[L])=(mid-L)/(H-L)* can be deduced, and *mid* can be obtained using *mid=L+R*(H-L)*.

The difference between the NFDT algorithm and the typical FSCS (Fixed Size Candidates Set) ART algorithm is that the next test case is determined based on the position of test cases previously executed by the NFDT algorithm. The input domain is divided into two areas based on the previously executed test cases, thereby reducing the search space and number of distance calculations. The improved binary search algorithm can help to identify the candidate test case closest to previously executed test cases. According to the distance between the closest and executed test cases, a decision is made as to whether or not distance calculations will be made for all the candidate test cases, thus potentially reducing the total number of distance calculations performed, similar to the filtering technique used by Chan et al [21].

(4) **CFDT Algorithm**

The CFDT algorithm uses the restricted adaptive random testing technique [22] to select the next test case, using an exclusion region radius. Generally speaking, the selected test cases have better detection capability for finding the security exceptions of Web services than general test cases. There are two reasons for this. Firstly, the selected test cases are always away from previously executed test cases that have been generated outside the exclusion region: more distant test cases can more easily find security exceptions than normal test cases [16]. Secondly, the selected test cases have been mutated based on mutation operators designed to detect special security exceptions.

Two parameters, *A* and *P*, are defined to measure the SOAP input domain, when it is a circle or an ellipse. *A* and *P* represent the area and perimeter of an ellipse, $A=\pi ab$, $P=\pi(3(a+b)/2-\sqrt{ab})$, respectively [23,24] (a and b are the radii of the ellipse; when a = b, the ellipse is a circle). *S* is the set of test cases to be tested; *C* is the set of test cases randomly generated; and N is the number of test cases in S. The first test case is randomly generated, and the subsequent ones are generated using an iterative approach [22]. A parameter $R=\sqrt{A/(2n\pi)}$ is used to determine the size of the exclusion region. Each test case in *S* is set as the center of a region, with *R* as the radius of the circular exclusion region. The first generated test case not falling in an excluded region is then selected as the next test case. An adjustment parameter *r* is introduced to compensate for the effects of overlapping zones and portions of zones lying outside the input domain. *R* is set as $\sqrt{Ar/(2n\pi)}$. The CFDT algorithm is shown in Algorithm 3.

Algorithm 3: CFDT
Input: the circle center $e_1(x, y)$ and radius *R* of SOAP message input region
Output: the set of test cases $S=\{e_1,e_2,...,e_n\}$
(1) **set** $S=\{\}, C=\{\}, n=0, r=1$;
(2) randomly generate e_1 by using related mutation strategies and operators and $S=S\cup e_1$;
(3) **while** (*R* !=0) **do**
(4) { find an exclusion circle (e_i (*i*=1,2,3....), $R=\sqrt{A/(2n\pi)}$), randomly generate *k* test cases {c_1, c_2, …., c_k} and then $\{c_1, c_2, ..., c_k\}\notin(e_i, R=\sqrt{A/(2n\pi)})$ $C=C\cup\{c_1,c_2,...,c_k\}$;
(5) sort the *k* test cases according to *x*-value from small to large, calculate all distances d_i, and then find the test case e_i whose distance is the largest and $S=S\cup e_i$, $n=n+1$;
(6) set *r* to adjust the exclusion region;
(7) }
(8) **return** the set of test cases $S=\{e_1, e_2,..., e_n\}$.

(5) **Weighted Ming distance**

If the number of SOAP parameters (*n*) is three or more, then the inputs are regarded as the n-tuple data set (*T*), with each $t=(x_1, x_2, ..., x_n)$, $t\in T$ being a single input from T. When a test case (*e*) is generated randomly, a new coordinate system is defined based on it, with each previously executed point (test case) translated appropriately. Without loss of generality, the following explanation of this method is in 2D, but the method applies to higher dimensions: Lines L_1 and L_2 are perpendicular axes through the point *e*, dividing the area that includes all points within the neighborhood of *e* into four sub-areas *M, N, S and O*. The four sub-areas are marked as the neighborhood areas of point *e*. Lines L_1 and L_2 are also seen as the boundaries between the four areas, with the corners formed by the lines being called neighborhood angles. Across from each neighborhood angle a diagonal is formed, enclosing the neighborhood area. Any points in the neighborhood area should be filtered using related algorithms [18]. Based on the neighborhood areas and some rules [18][25], the weighted Ming distance (WD) between a point t and e is defined as $WD=(\sum_{i=1}^{n}|x_i-y_i|^2\times w_i)^{1/2}/\sum_{i=1}^{n}w_i$, where w_i represents the corresponding weight for every input parameter to define the contribution of different parameter. The formula can measure the distance between different inputs. Given a current test case (*e*), the Furthest Neighbor (FN) formula is used to select the next test case, and is defined as $FN(e)=\{r\in T\mid\forall t\in T: WD(e,r)\geq WD(t,r)\}$. The formula guarantees that the distance between the current and next test case is always greater than or equal to the distance between the next test case and any test case of T.

(6) **Inverse probability distribution**

If the *n*-tuple parameters are from a continuous input space and the inverse probability distribution function for the input space can be obtained, then it can be used to guide generation of some

unconventional test cases to detect security exceptions. Generally speaking, unconventional inputs can effectively trigger the security exceptions for Web services. The input distribution function is usually a probability density function, whose output ranges from 0 to 1, where 0 means that it is impossible to select inputs from the input domain, and 1 means that the inputs from the input domain are 100% available. The main steps needed to get the inverse probability distribution function are as follows [26].

Step 1: Describe the probability of each input (an ordered n-tuple) as a value in the $n+1^{th}$ dimension;

Step 2: Determine the hyper-plane which is defined by setting the $n+1$ dimension value to a constant, $1/K$, where K is the cardinality of the input space;

Step 3: Reflect the input distribution through this hyper plane;

Step 4: If any of the resulting values in the $n+1^{th}$ dimension are negative, translate the graph by a vector of magnitude C, so that all the values in the $n+1$ dimension are non-negative;

Step 5: Normalize the resulting graph in $n+1$ space, dividing each value by the total volume. At the end of this step, the value in $n+1$ space associated with each n-tuple is the probability of selection in the inverse probability distribution function.

The SOAP message is obtained by parsing the WSDL file of the Web services being tested, and is then transformed into a DOM tree. Based on the number and type of SOAP parameters, the appropriate TCFN algorithm is called to generate test cases. The complexity of the TCFN algorithm is mainly determined by the BRA algorithm, ResStr, NFDT, CFDT, weighted Ming distance and inverse probability distribution algorithms. In the BRA algorithm, flipping all bits (from 0 to 1, and 1 to 0) is time consuming. If the bit length of the integer is n, then the complexity of BRA algorithm is $O(n)$. In the ResStr algorithm, traversing the entire string is time consuming. If the length of the string is n, then the complexity of ResStr algorithm is $O(n)$. In the NFDT algorithm, a set of test case candidates randomly generated in the input domain is maintained. Each time a new test case is required, the candidate test case that is farthest from all previously executed test cases is selected. The runtime of the NFDT algorithm when generating n test cases is in the order of $O(n^2)$. The main time cost of the CFDT algorithm is the large number of distance calculations which are performed when new test cases are selected. The runtime of the CFDT algorithm when generating n test cases is in the order of $O(n^2 \log n)$. The time complexity of both weighted Ming distance algorithm and inverse probability distribution algorithm is respectively $O(n^2)$. The total time complexity of the TCFN algorithm is therefore $O(n)+O(n)+O(n^2)+O(n^2)+O(n^2)+O(n^2 \log n) = O(n^2 \log n)$.

V. EXPERIMENT AND ANALYSIS

A. *Experimental implementation*

To investigate and evaluate the proposed TCFN algorithm, a Web service vulnerability testing system (WSVTS) was implemented. The WSVTS framework is shown in Figure 2. WSVTS obtains the interface information by parsing the uniform resource locator (URL) of the Web service, and gets the SOAP message by parsing the WSDL document.

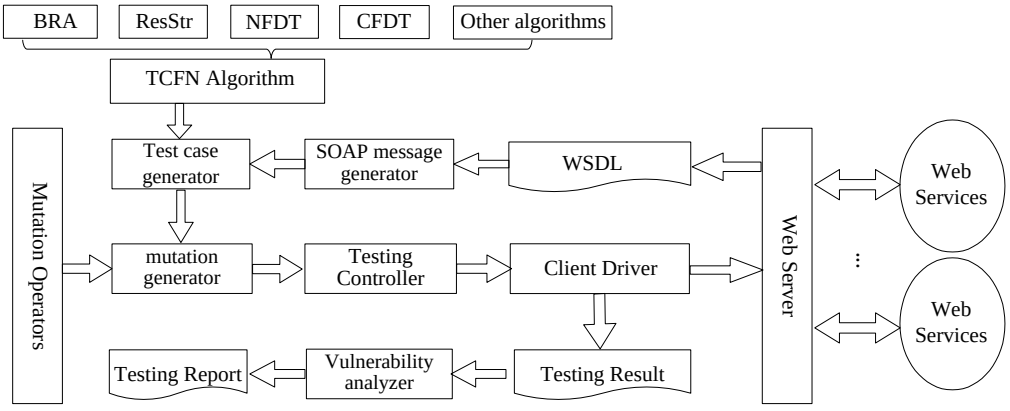

Figure 2. The WSVTS framework

WSVTS was implemented in Visual C# on the Microsoft.NET platform, and contains four main function modules: (a) the SOAP message generator; (b) the SOAP message mutation generator; (c) the test case generator; and (d) the Web service vulnerability analyzer. The details of these major modules are presented in the following.

1) SOAP message generator

The input to the SOAP message generator is a WSDL file of the Web service being tested, and consists of the response message data type, the transmission protocol and the Web service address information. The output is a Web service SOAP message.

2) SOAP message mutation generator

Based on mutation operators designed for different fault types, the mutation module mutates the SOAP message parameter type and value. The parameter type and number are obtained from the SOAP message generator, and the test cases are obtained from the test case generator.

3) Test case generator

The test case generator provides a convenient interface for the tester to input test cases, and can also use different algorithms based on the SOAP message parameter number, as analyzed by the SOAP message generator.

4) Vulnerability analyzer

The vulnerability analyzer generates a vulnerability report after testing the Web services. It analyzes the Web service vulnerability based on the security specifications, and reports the number of security exceptions and faults found.

As can be seen in the WSVTS flow chart (Figure 3), the SOAP message is obtained by parsing the WSDL file of the Web services being tested. Then, using an XML analysis technique, the number and type of SOAP message parameters are extracted, based on which, the appropriate TCFN algorithm is called to generate test cases. The Web services are tested based on the testing controller and client driver, using the generated test cases. Finally, the vulnerability testing report is obtained based on observations of the response messages received from the client of the Web services being tested.

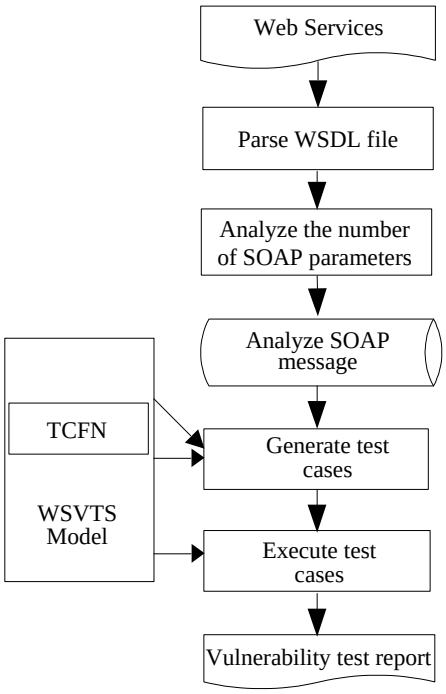

Figure 3. Flow chart of the Web service vulnerability testing system

In the experiments, in addition to several open Web services, some specifically written services were also analyzed. The list of tested Web services is shown in Table II.

During the experimental process, the function of the IPO mutation operator was merged with that of either the IIV or PFB mutation operator to generate test cases, according to the specific circumstances and SOAP message types. Different mutation operators may find the same error for the same Web service, in which case the error was counted only once. Similarly, the same fault found by different test cases

generated by the same perturbation operator was also only counted once. The operator efficiency (*OE*) defines the efficiency of an operator in terms of finding faults, and is calculated as *OE=EF/TC*, where *EF* is the number of faults found and *TC* is the total number of test cases generated by the operators. The efficiency of the mutation operators is shown in Figure 4. Different mutation operators have different efficiency, with the FVS operator having the highest (36.52%).

B. Experimental results and analysis

Web service vulnerabilities were found by the proposed approaches. Although the proposed mutation operators are applicable to related approaches, the test case generation rules may differ. Also, the continuous types of test case generation are more complex than the discrete types, and test cases for the continuous types can be adapted to the discrete types but not vice versa. We next compare our proposed approach with two others, SOAPUI [27] and SMAT-WS [7].

1) Comparison of WSVTS and SOAPUI

A total of 20 kinds of specially designed Web services were investigated using the two approaches based on the SOAP message parameter type. The SOAPUI [27] is an open source Web service testing tool, and WSVTS is a testing tool based on the approach proposed in this paper. Table III shows the experimental results for the open source tool SOAPUI, in which the test cases are manually entered according to the SOAP message parameter type; and Table IV shows the results for WSVTS. Based on these results, the overall efficiency (OE) of the mutation operators generated by the two approaches are calculated to be approximately 21.1% and 23.7%, respectively, confirming the feasibility of our proposed approach, and the validity of the test cases generated.

Table II. The tested Web services

No.	Service Name	The number of service methods	The number of method parameters	Description
WS1	Stock	8	23	Searching stock information
WS2	Weatherforecast	7	19	Weather forecast service
WS3	E-Banking	9	25	Online banking service
WS4	Bookfinding	6	15	Searching book information
WS5	Domainfinding	5	13	Searching domain and IP address
WS6	Petinformation	7	16	Searching Pet information
WS7	Traintime	7	14	Searching train timetable
WS8	Planetime	5	12	Searching aircraft flight information
WS9	QQcheckonline	7	13	Searching QQ online information
WS10	Queryresults	9	22	Searching student achievement information
WS11	Producedorder	8	16	Searching production order information
WS12	Calculator	7	15	Arithmetic calculating service
WS13	Maxdivisor	5	10	Finding the greatest common divisor of two numbers
WS14	Mod	4	8	Finding the remainder of two numbers
WS15	Reversestring	8	14	Reversing the string
WS16	Stringcopy	6	12	Copying the string
WS17	Stringlength	4	8	Obtaining the length of string
WS18	Login	5	8	User login
WS19	Vote	5	16	Getting the result of the vote
WS20	Echoinformation	6	13	Echoing personal information

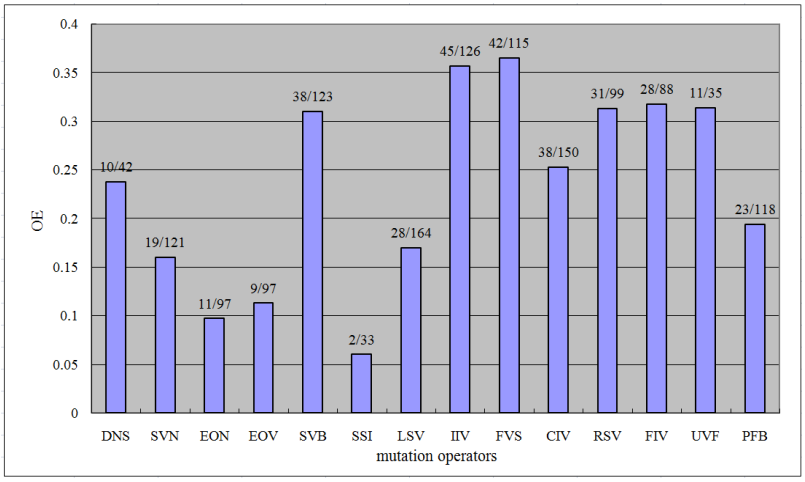

Figure 4. Efficiency of the mutation operators

Table III. Test results of the SOAPUI tool

Mutation operators	**DNS**	**SVN**	**EON**	**EOV**	**SVB**	**SSI**	**LSV**	**IIV**	**FVS**	**CIV**	**RSV**	**FIV**	**UVF**	**PFB**	**Total**	**OE**
Number of test cases generated	42	124	113	97	130	40	211	151	115	197	98	102	41	118	1579	21.1%
Faults found	8	19	11	9	38	2	28	45	42	36	31	28	7	30	334	

Table IV. Test results of WSVTS tool

Mutation operators	**DNS**	**SVN**	**EON**	**EOV**	**SVB**	**SSI**	**LSV**	**IIV**	**FVS**	**CIV**	**RSV**	**FIV**	**UVF**	**PFB**	**Total**	**OE**
Number of test cases generated	42	118	113	97	123	33	164	126	115	150	99	88	35	118	1421	23.7%
Faults found	10	19	11	11	38	2	28	45	42	38	31	28	11	23	337	

Figure 5 gives a comparison of the efficiency of the two approaches, showing that for most operators, the number of faults found by the WSVTS approach is higher than that found by the SOAPUI tool (exceptions being the EON, FVS, RSV, and PFB operators). The UVF operator appears particularly efficient. The faults found consist of some common vulnerability faults such as memory leak, buffer overflow, cross-boundary access, and arithmetic security faults including dividing by zero and out-of-range operand values. Thus, the designed operators and our approach are confirmed to be very effective.

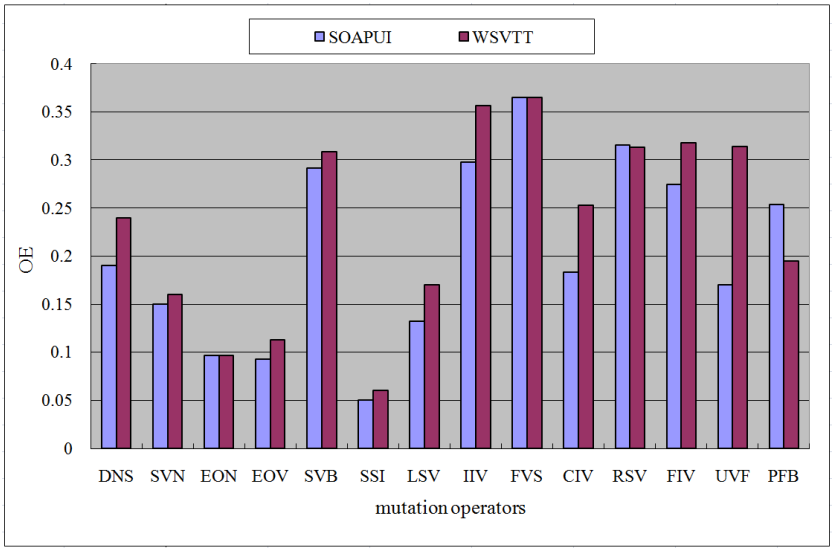

Figure 5. Comparison of the WSVTS and SOAPUI efficiencies

2) Comparison of SMAT-WS, WSVTS, and SOAPUI

Research on SOAP message mutation testing is still not common. The experimental results of SOAP message perturbation reported by Almeida & Vergilio [7] is reproduced here in Table V. Their proposed mutation operators are different from ours because of the different Web services, therefore we compare the approaches based on the overall efficiency of the mutation operators: the overall effectiveness of the test cases generated by the SMAT-WS testing tool is 15.7%. A comparison of all three methods is shown in Figure 6.

Table V. SMAT-WS test results [7]

Mutation operators	**I**	**N**	**BE**	**IN**	**VI**	**S**	**B**	**U**	**ML**	**Total**	**OE**
Number of test cases generated	54	54	363	43	45	54	162	54	108	937	15.7%
Faults found	16	21	24	2	8	19	27	16	15	148	

Mutation Operators: Incomplete (I), Null (N), Boundary Extension (BE), Inversion (IN), Value Inversion (VI), Space(S), Unauthorized (U), Mod_Len(ML), Boundary(B)

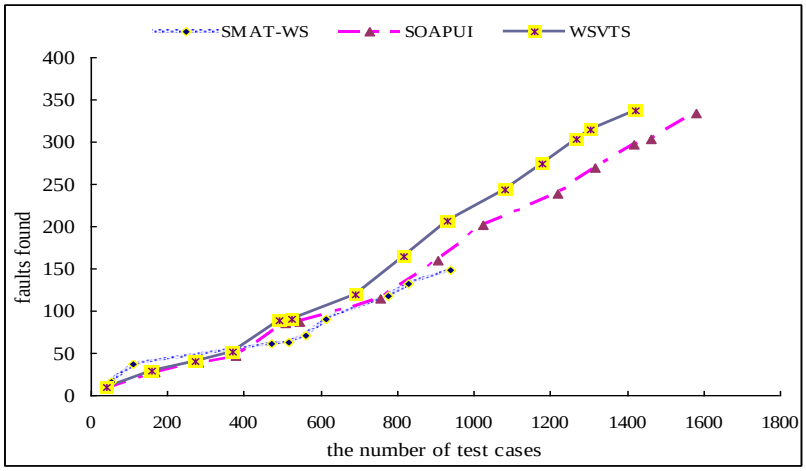

Figure 6. Comparison of the SMAT-WS, SOAPUI, and WSVTS tools

The experimental results in Figure 6 show that SMAT-WS finds more faults at the earlier stage of testing, but that the rate of faults found by WSVTS increases fastest, supporting the validity of this approach. The fault-finding abilities of the SMAT-WS and SOAPUI approaches are similar within a certain range. Although the three approaches (SMAT-WS, SOAPUI and WSVTS) are all based on SOAP message mutation, the corresponding proposed mutation operators are different because the Web services tested with SMAT-WS are different from those tested by SOAPUI and WSVTS. In general, the number of test cases generated is different because of the different mutation operators applied to different situations as well as the number of faults.

Compared with the other methods, the advantages of the WSVTS tool include that the mutation operators expand according to the characteristics of the SOAP message in the experiment – in other words, the testing is more comprehensive; and the algorithm is automatically called to generate test cases according to the number of parameters and the SOAP message type. The targeted faults consist of buffer overflow faults, cross-boundary access faults and arithmetic security faults.

VI. CONCLUSIONS

Research on Web service vulnerability testing remains limited, partly due to their cross-platform and differing characteristics. In this paper we have presented mutation operators designed for SOAP messages, and a mutation testing algorithm for the automated generation of test cases.

By designing appropriate SOAP message mutation operators, the security of the Web services can be tested from the client side, and vulnerability faults can be identified from the user perspective. In most cases, compared with the classic farthest neighbor algorithm, the proposed TCFN algorithm reduces the number of distance calculations. Compared with the pure random testing, the proposed TCFN algorithm can detect more faults with fewer test cases. Because specifically tailored test cases can be generated, the efficiency and quality of test case generation can be improved. Furthermore, the test cases can also be generated

automatically, using legal and illegal input parameters and mutation operators. The effectiveness of the proposed approach has been shown to be higher than that of other available approaches. The efficiency of the proposed mutation operators is higher than other approaches such as SMAT-WS. In addition, the approach can detect more vulnerability faults than other approaches with the same test cases.

In the future, we would like to continue research in the following areas: firstly, we will do more experiments to verify the reliability of the proposed approaches. Secondly, we will research how to further reduce the redundant test cases after mutating. Thirdly, the automatic process of test case generation and mutation also need to be further improved to enhance the testing efficiency.

ACKNOWLEDGMENTS

We would like to thank T. Y. Chen for his insightful discussion and suggestions. This work is partly supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61202110 and Natural Science Foundation of Jiangsu Province under Grant No. BK2012284.

REFERENCES

[1] S. Hanna, M. Munro, An approach for wsdl-based automated robustness testing of web services. The 16th International Conference on Information Systems Development (ICISD'2009), Springer, 2009, pp.1093-1104.

[2] T. Takase, K. Tajima, Efficient Web Service Message Exchange by SOAP Bounding Framework. The 11th IEEE International Enterprise Distributed Object Computing (EDOC'2007), IEEE Computer Society, 2007, pp.63-72.

[3] L.Wu, X.K. Li, H. Wang. Research on the Reliability Testing of Web Service Based on Fault Injection Technology. Journal of Chinese Computer System, 2007, 28(1):127-131. (in Chinese)

[4] M. Palacios, J. Garcia-Fanjul, J. Tuya, Testing in service oriented architectures with dynamic binding: a mapping study. Information and Software Technology, 2011, 53(3): 171–189.

[5] C.A. Sun, G. Wang, B.H. Mu, H. Liu, Z.S. Wang, T. Y. Chen, A Metamorphic Relation-Based Approach to Testing Web Services Without Oracles. International Journal of Web Services Research, 2012, 9(1): 51-73.

[6] C.A. Sun, G. Wang, B.H. Mu, H. Liu, Z.S. Wang, T. Y. Chen, Metamorphic testing for web services: framework and a case study. The IEEE International Conference on Web Services (ICWS'2011), IEEE Computer Society, 2011, pp. 283-290.

[7] L.F. de Almeida, S.R. Vergilio, Exploring Perturbation Based Testing for Web Services. The IEEE International Conference on Web Services (ICWS'2006), IEEE Computer Society, 2006, pp.717-726.

[8] H. C. Kim, Y. H. Choi, D. H. Lee, Efficient File Fuzz Testing Using Automated Analysis of Binary File Format. Journal of Systems Architecture, 2011, 57(3):259-268.

[9] S. Bekrar, C. Bekrar, R. Groz, L. Mounier, Finding Software Vulnerabilities by Smart Fuzzing. Proceeding of the Fourth IEEE International Conference on Software Testing, Verification and Validation (STVV'2011), IEEE Computer Society, 2011, pp. 427-430.

[10] J. Offutt, W. Xu, Generating Test Cases for Web Services Using Data Perturbation. ACM SIGSOFT Software Engineering Notes, 2004, 29(5):1-10.

[11] A. C.V. de Melo, P. Silveira, Improving data perturbation testing techniques for Web Services. Information Science, 2011,181 (03):600-619.

[12] P. Silveira, A. C. V. de Melo, Exploring XML Perturbation Techniques for Web Services Testing. International Conference on Web Engineering (ICWE'2009), Spinger LNCS, 2009, v. [5648]:355-369.

[13] J.F. Chen, Q. Li, C.Y. Mao, D. Towey, Y.Z. Zhan, H.H. Wang: A Web services vulnerability testing approach based on combinatorial mutation and SOAP message mutation. Service Oriented Computing and Applications, 2014, 8(1):1-13.

[14] L. Novak, A. Zamulin, A formal model for XML schema. Proceedings of the 21st International Conference on Data Engineering Workshops (ICDEW'2005), IEEE Computer Society, 2005, pp.1283-1293.

[15] W. Xu, J. Offutt, J. Luo, Testing Web Services by XML Perturbation. Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'2005), IEEE Computer Society, 2005, pp.257-266.

[16] J.F. Chen, Y.S. Lu, X.D. Xie, Component Security Testing Approach by Using Interface Fault Injection. Journal of Chinese Computer System, 2010, 31(6):1090-1096. (in Chinese)

[17] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. McMinn, An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 2013, 86(8), 1978-2001.

[18] C. Böhm, S. Berchtold, and D. A. Keim, Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys, 2001, 33(3): 322-373.

[19] T. Y. Chen, F.-C. Kuo, R. G. Merkel and T. H. Tse, Adaptive Random Testing: the ART of Test Case Diversity. Journal of Systems and Software, 2010, 83(1):60-66.

[20] M. H. Alsuwaiyel, Algorithms: Design Techniques and Analysis. Publisher: World Scientific Pub Co Inc, November 1998.

[21] K. P. Chan, T. Y. Chen, and D. Towey, Adaptive Random Testing with Filtering: An Overhead Reduction Technique. The 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05), Taipei, Taiwan, China, IEEE Computer Society, 2005, pp. 292-299.

[22] K. P. Chan, T. Y. Chen and D. Towey, Restricted Random Testing: Adaptive Random Testing by Exclusion. International Journal of Software Engineering and Knowledge Engineering, 2006, 16(4):553-584.

[23] T.Y. Chen, F.-C. Kuo, C.A. Sun, Impact of the Compactness of Failure Regions on the Performance of Adaptive Random Testing. Journal of Software, 2006, 17(12):2438-2449.

[24] I.N. Bronshtein, K.A. Semendyayev, G. Musiol and H.Mühlig, Handbook of Mathematics. Publisher: Springer, 5th edition, October 2007.

[25] B.H. Li, Z.X. Hao. Efficient Filtration and Query Algorithm of Reverse Furthest Neighbor. Journal of Chinese Computer Systems, 2009, 30 (10): 1948-1951. (in Chinese)

[26] J. M. Voas, K. W. Miller, Predicting software's minimum-time-to-hazard and mean-time-to-hazard for rare input events. The Sixth International Symposium on Software Reliability Engineering (ISSRE'1995), IEEE Computer Society, 1995, pp.229-238.

[27] SoapUI, SmartBear Software, available at http://www.soapui.org (last access September 2012)